



Programación Python para Machine Learning

Lección 2: Estadística descriptiva,
visualización y preparación de datos.

ÍNDICE

1. Entendiendo los datos	2
1.1. Un vistazo a los datos	3
1.2. Estadísticos descriptivos.....	4
1.3. Distribución por clases.....	5
1.4. Correlación entre atributos.....	5
1.5. Sesgo de la distribución de los datos	6
2. Visualización de datos	7
2.1. Gráficos univariantes	7
2.2. Gráficos multivariantes.....	11
3. Preprocesamiento de datos	14
3.1. Necesidad de preprocesar los datos	14
3.2. Transformaciones de datos	15
4. Puntos clave.....	19

Lección 2: Estadística descriptiva, visualización y preparación de datos.

1. ENTENDIENDO LOS DATOS

El primer paso a la hora de obtener los mejores resultados en cualquier proyecto de Machine Learning es conocer y comprender los datos con los que se está trabajando. Completar tareas como visualizar los datos en bruto, conocer las dimensiones y el tipo de atributos del conjunto de datos, analizar el balanceo de clases, descubrir las relaciones entre sus variables o ver el sesgo de las distribuciones de cada atributo, representa un punto de partida clave en tal proceso. Para ello, contamos con la ayuda de la estadística descriptiva, una disciplina que proporciona las técnicas para recoger, presentar y caracterizar las variables de un conjunto de datos mediante sus parámetros básicos, tablas o gráficos.



Objetivos

- | Entender los datos para obtener el máximo rendimiento de ellos
- | Utilizar las técnicas de estadística descriptiva para resumir los datos.
- | Analizar las relaciones presentes en los datos, numérica y gráficamente.
- | Conocer los principios y saber aplicar las técnicas de preprocesamiento de datos.

1.1. Un vistazo a los datos

No hay nada que sustituya a la observación de los datos en bruto. Examinar los datos en crudo puede revelar aspectos que no se pueden obtener de ninguna otra manera. También puede dar una idea sobre cómo preprocesar y manejar mejor los datos para las tareas posteriores en el proceso de Machine Learning. En Python, se pueden examinar las primeras filas de sus datos utilizando el método *head()* del DataFrame de Pandas. La primera columna muestra el número de fila, lo que es útil para referirse a un patrón específico.

Otro aspecto importante a conocer es la cantidad de datos que se tienen, tanto en términos de filas como de columnas. Demasiadas filas (instancias, patrones, puntos, observaciones) pueden provocar que los algoritmos tarden un excesivo tiempo de entrenamiento. Un número demasiado pequeño de ellos pueden no ser suficientes para completar el entrenamiento de los modelos. En el caso de las columnas (características, dimensiones, atributos, campos), el exceso de las mismas puede hacer que los algoritmos se distraigan, no converjan o el modelo resultante tenga un mal rendimiento. Para conocer la forma y el tamaño de su conjunto de datos, el Pandas DataFrame dispone de la propiedad *shape*.



Conceptos

Los términos: instancias, patrones, puntos, observaciones, registros, filas... se refieren conceptualmente a lo mismo, cada uno de los puntos de los que se disponen para hacer un análisis.

De manera análoga, los términos: características, factores, dimensiones, variables, atributo, propiedad, campo, columnas... son los atributos que describen cada una de las instancias del conjunto de datos.

Conocer el tipo de cada atributo supone otro factor importante y determinante. Es necesario diferenciar adecuadamente las variables numéricas, categóricas y ordinales. Un modo para conocer los tipos de atributos es echando un vistazo a los datos, como se ha explicado anteriormente mediante la visualización de los primeros registros. Sin embargo, el modo más adecuado es enumerar los tipos de datos utilizados por el DataFrame para caracterizar cada atributo utilizando la propiedad *dtypes*.

1.2. Estadísticos descriptivos

Los estadísticos descriptivos pueden aportar una visión de la conformación de cada atributo. Concretamente, el método *describe()* de DataFrame lista una serie de propiedades estadísticas de cada variable:

- | Conteo.
- | Media.
- | Desviación típica.
- | Valores máximo y mínimo.
- | Q1, Q2 (mediana) y Q3.

Se puede formatear la salida de *describe()* mediante *pandas.set_option()* para limitar la precisión o la anchura de cada elemento y que la tabla gane en legibilidad.

Una vez vemos los parámetros de los datos mediante este método, merece la pena tomarse un tiempo para revisar los resultados. En particular, es interesante analizar la presencia de valores NA para datos faltantes o distribuciones sorprendentes de los atributos.

1.3. Distribución por clases

Para problemas de clasificación es de vital importancia conocer lo equilibrados que están el número patrones por clases. Los problemas muy desequilibrados, es decir, muchas más observaciones para una clase que para otra, son bastante comunes y necesitan de un tratamiento específico en la fase de preparación de los datos del proyecto. Es posible obtener una idea de la distribución del atributo clase en un DataFrame de Pandas. Para ello habría que agruparlo por la columna que determina la clase y calcular su tamaño del siguiente modo:

```
# Distribución de clases
from pandas import read_csv
filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)
class_distribution = data.groupby('class').size()
print(class_distribution)
```

1.4. Correlación entre atributos

La correlación es el término que se refiere a la relación entre dos variables y cómo pueden o no cambiar al mismo tiempo. El método más común para calcular la correlación entre dos variables numéricas es mediante el Coeficiente de Correlación de Pearson, que supone una distribución normal de los atributos implicados. Una correlación de -1 o 1 muestra una correlación totalmente negativa o positiva, respectivamente, entre dos variables.

Por el contrario, un valor de 0 muestra que no hay ninguna correlación entre ambas. Algunos algoritmos de Machine Learning, como la regresión lineal y logística, presentan un mal rendimiento si hay atributos muy correlados en el conjunto de datos.

Por lo tanto, es una buena idea revisar todas las correlaciones por pares de los atributos en su conjunto de datos. Para obtener una matriz de correlación de las variables de un dataset, el DataFrame de Pandas ofrece el método *corr()*.

La matriz resultante es simétrica, enumera todos los atributos en la parte superior y en el lateral derecho y especifica la correlación entre todos los pares de atributos. La diagonal principal, evidentemente, muestra la correlación perfecta de cada atributo consigo mismo.

1.5. Sesgo de la distribución de los datos

Muchos algoritmos de Machine Learning asumen una distribución gaussiana de los datos. Conocer el hecho de que un atributo tiene un sesgo será útil a la hora de realizar una preparación de los datos para corregir tal sesgo y mejorar posteriormente el rendimiento de los modelos. Puede calcular el sesgo de cada atributo utilizando el método `skew()` de `DataFrame`. Un valor claramente mayor que cero evidencian una distribución de datos sesgada por la derecha. Los valores negativos, indicarán una variable sesgada por la izquierda. En el caso de obtener para una variable un valor igual o cercano a cero, se muestra una variable con distribución normal.

2. VISUALIZACIÓN DE DATOS

Como se ha mencionado en la sección anterior, comprender los datos es un paso determinante a la hora de obtener el mejor rendimiento de los algoritmos de Machine Learning. De este modo, la forma más rápida de adquirir un conocimiento inicial sobre los datos es utilizar técnicas visualización de datos.

2.1. Gráficos univariantes

Existen una serie de técnicas que se pueden utilizar para entender cada atributo de su conjunto de datos de forma independiente:

- | Histogramas.
- | Gráficos de densidad.
- | Gráficos de cajas y patas (boxplot).

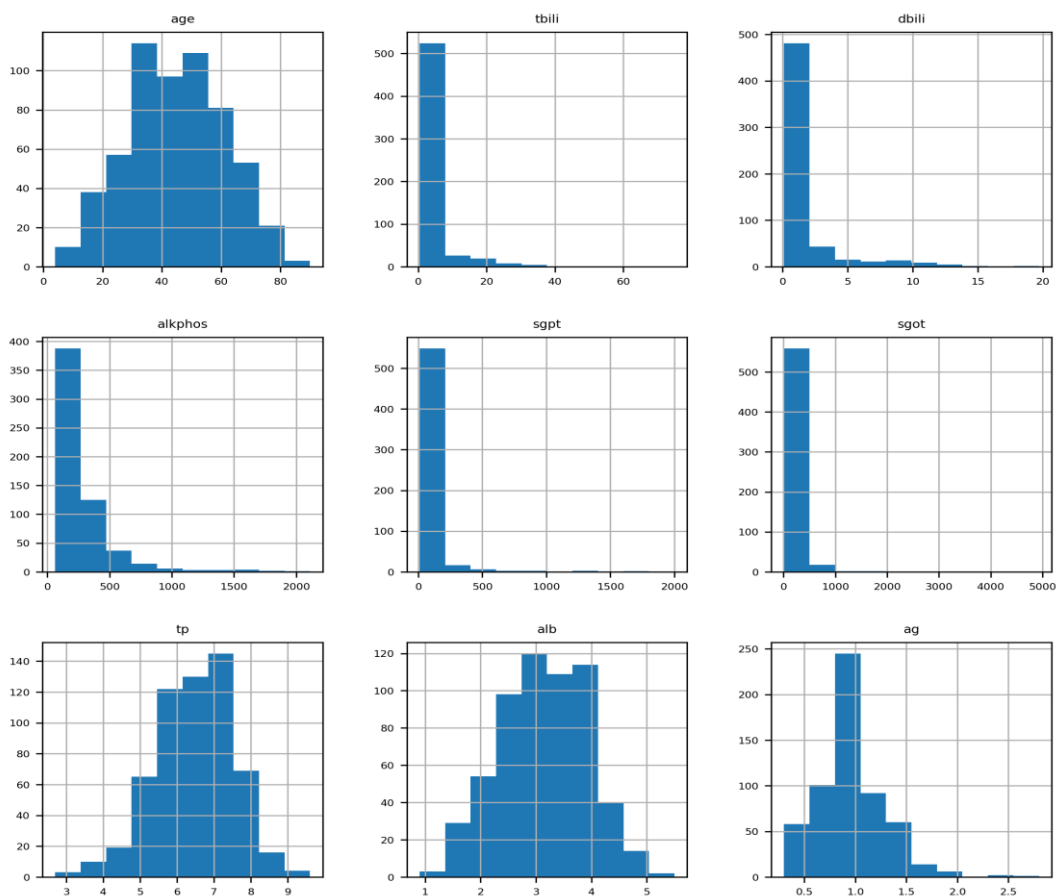


Ilustración 1: Histograma de cada atributo Indian-Liver-Patient.

Histogramas y gráficos de densidad

La función de distribución de las variables de una instancia es algo a tener en cuenta en Machine Learning puesto que hay métodos que suponen que las variables de entrada siguen una distribución normal.

En caso de no seguirla, se hace necesario realizar alguna transformación de la variable. Una forma rápida de hacerse una idea de la distribución de cada atributo es observar su histograma. Los histogramas son elementos gráficos, concretamente, gráficos de barras, que agrupan los datos en intervalos y proporcionan un recuento del número de observaciones en cada intervalo.

En la forma de los intervalos se puede obtener rápidamente una idea de si un atributo es gaussiano, asimétrico o incluso tiene una distribución exponencial.

También puede ser de gran utilidad a la hora de detectar posibles valores atípicos (outliers). El método *hist()* del Pandas DataFrame es el adecuado para generar un histograma de los datos, bien sea para una sola variable (numérica o categórica) de un conjunto de datos o para todos los atributos (numéricos) del mismo.

```
# Gráficos univariantes
from matplotlib import pyplot
from pandas import read_csv

filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)

input_data=data[data.columns[:-1]]
input_data.hist()
input_data.plot(kind='density', subplots=True, layout=(3,3),
sharex=False)
input_data.plot(kind='box', subplots=True, layout=(3,3),
sharex=False,sharey=False)
pyplot.show()
```

En la **¡Error! No se encuentra el origen de la referencia.** se puede ver que los atributos *age*, *tp*, *alb* y *ag* quizá siguen una distribución normal. Las distribuciones del resto de atributos posiblemente sigan una exponencial. Advertir de la ausencia del atributo *gen* debido a que `hist()` directamente no grafica los atributos categóricos.

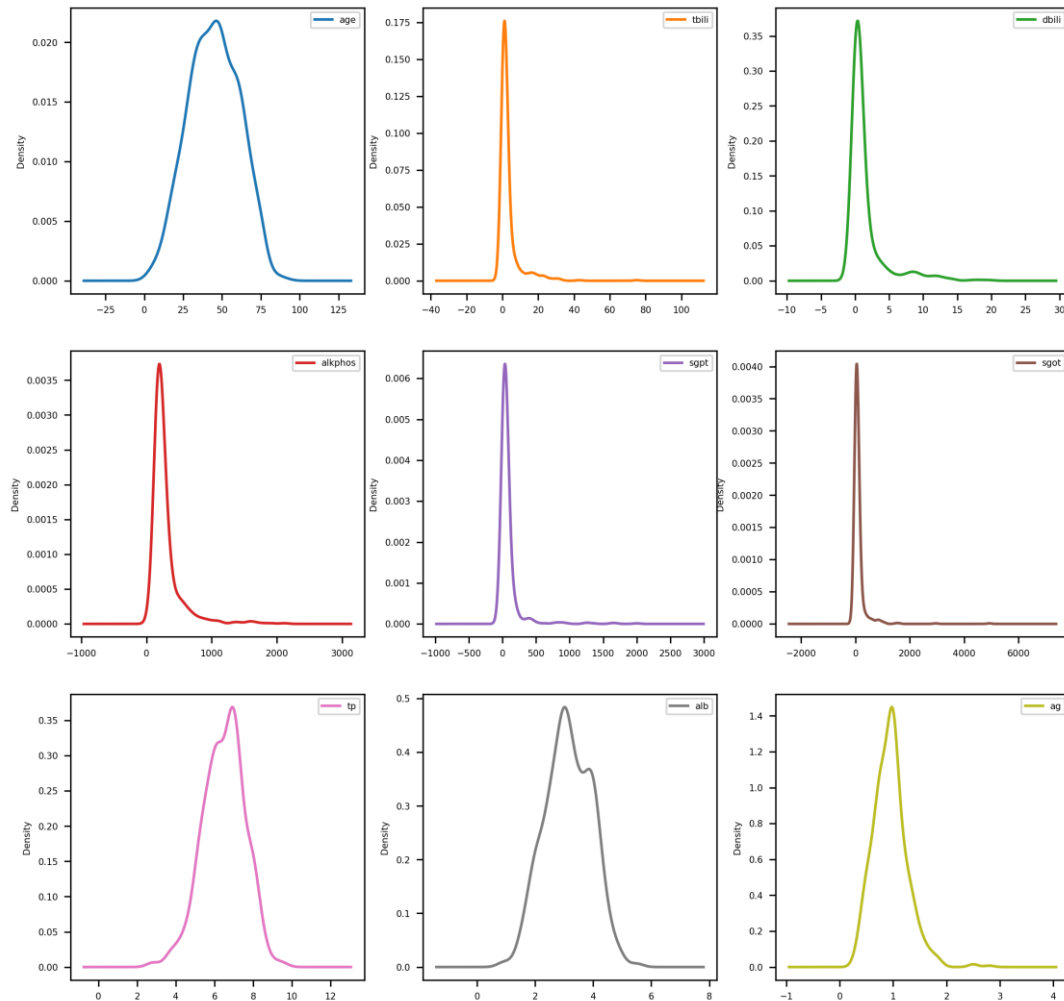


Ilustración 2: Gráficos de densidad por atributo del dataset Indian-Liver-Patient.

Los gráficos de densidad son otra forma de hacerse una idea rápida de la distribución de cada atributo. Como se ve en la Ilustración 2, los gráficos se parecen a un histograma abstracto con una curva suave dibujada a través de la parte superior de cada casilla. Esta gráfica ha sido obtenida mediante el método `plot(kind='density')` de la clase `DataFrame` de Pandas.

Gráficos de cajas y patas

Otra forma útil de revisar la distribución de los atributos es utilizar los gráficos de caja y cajas o boxplots. Los gráficos de caja resumen la distribución de cada atributo, dibujando una línea para una línea para la mediana y una caja alrededor de los cuartiles Q1 y Q3 (percentiles 25 y 75).

Como se observa en la Ilustración 3, obtenida utilizando el método `plot(kind='box')` de DataFrame, las patas dan una idea de la dispersión de los datos y los puntos fuera de las patas muestran los valores atípicos, valores que son 1,5 veces mayores que el tamaño del rango intercuartílico. Los extremos de las patas identifican los valores más extremos que no son considerados atípicos.

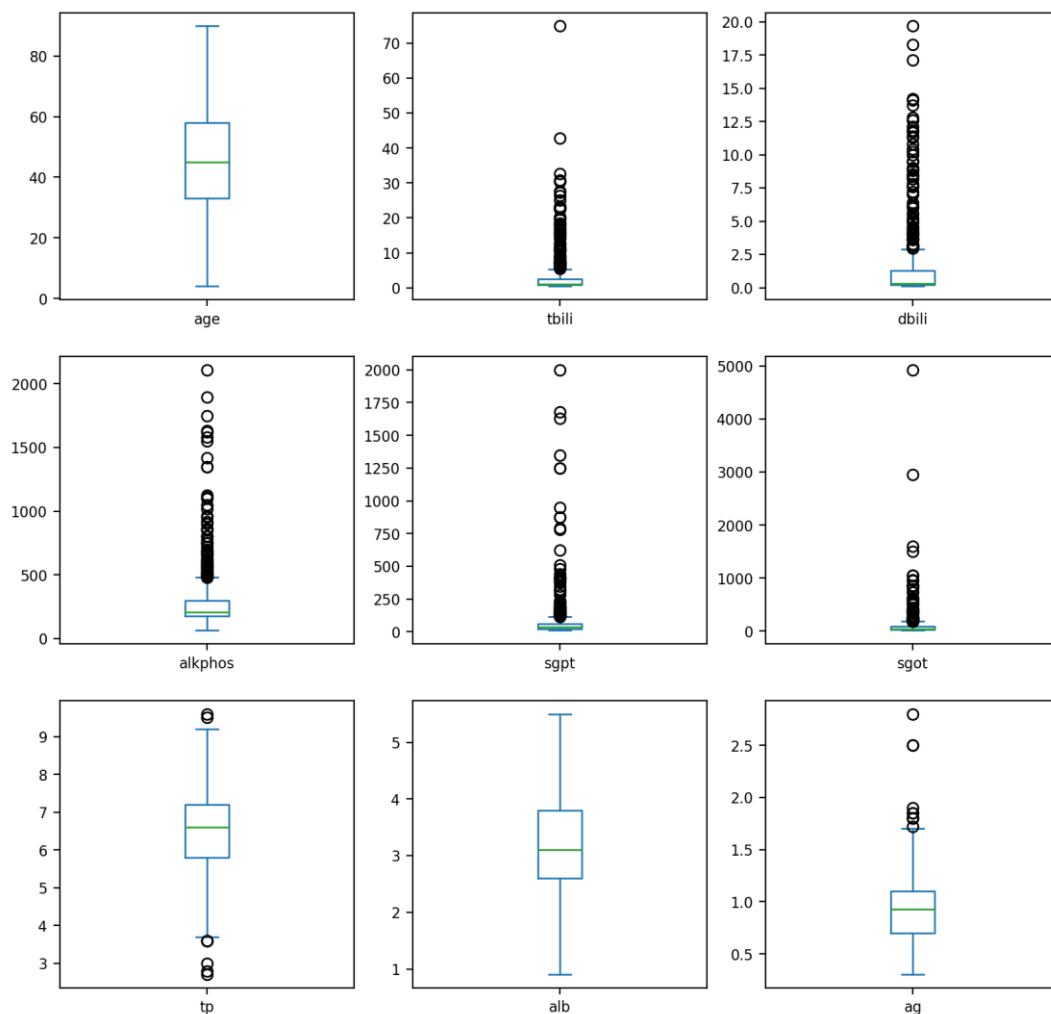


Ilustración 3: Boxplot para los atributos del dataset Indian-Liver-Patient.

2.2. Gráficos multivariantes

Para visualizar las interacciones entre múltiples variables en un conjunto de datos, se pueden utilizar los siguientes gráficos:

- | Gráfico de matriz de correlación.
- | Matriz de dispersión.

Representación gráfica de la matriz de correlación

La correlación da una indicación de cómo están relacionadas dos variables. El concepto de matriz de correlación fue explicado en la sección 1.4. Atender este aspecto puede ser interesante porque algunos algoritmos de Machine Learning, como la regresión lineal o la regresión logística, tienen un mal rendimiento si hay variables de entrada altamente correlacionadas en sus datos. Es práctico valerse de este gráfico genérico como primer paso para entender las correlaciones en un conjunto de datos, y luego leer unos datos numéricos más específicos si es necesario en la propia matriz.

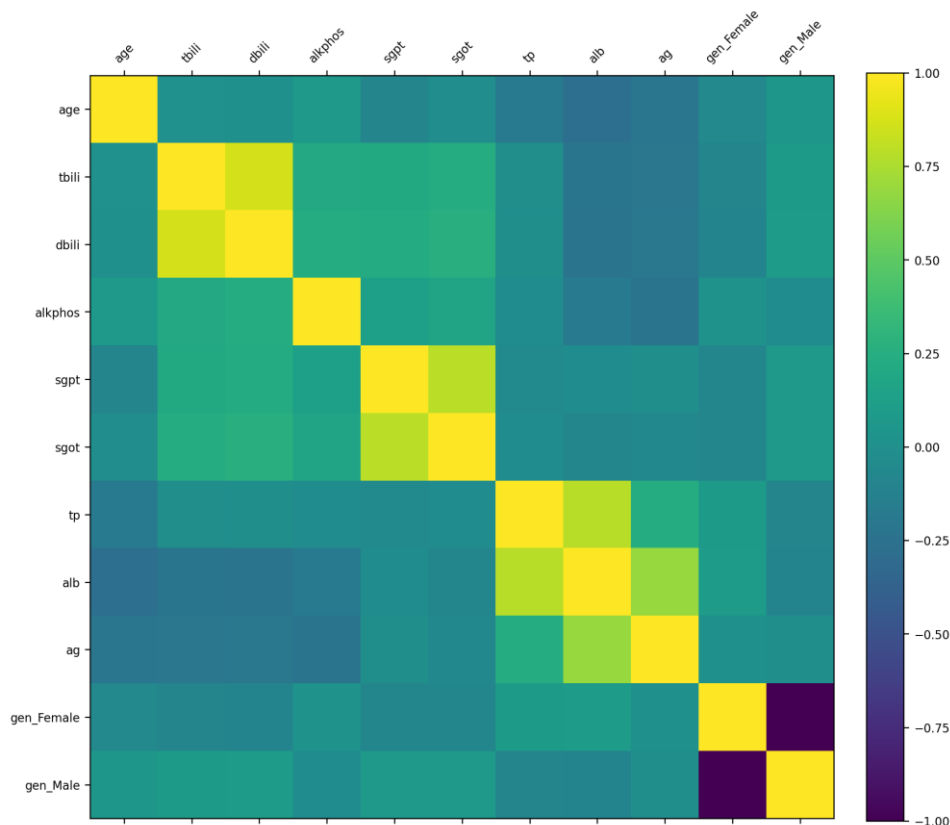


Ilustración 4: Matriz de correlación de las variables del dataset Indian-Liver-Patient.

Representación gráfica de la matriz de dispersión

Un gráfico de dispersión muestra la relación entre dos variables como puntos en dos dimensiones, un eje para cada atributo. Se puede crear un gráfico de dispersión para cada par de atributos de sus datos. La figura resultante de todos estos gráficos de dispersión juntos se denomina matriz de gráficos de dispersión. Los gráficos de dispersión son útiles para detectar relaciones estructuradas entre variables, como si se pudiera resumir la relación entre dos variables con una línea. Los atributos con relaciones estructuradas pueden estar correlacionados y ser buenas candidatas para ser eliminadas del conjunto de datos.

Al igual que la matriz de correlación anterior, la matriz de dispersión es simétrica. Esto es útil para observar las relaciones entre pares desde diferentes perspectivas. Como no tiene mucho sentido de incluir un gráfico de dispersión de cada variable consigo misma, la diagonal muestra bien los histogramas de cada atributo o bien la función de densidad.

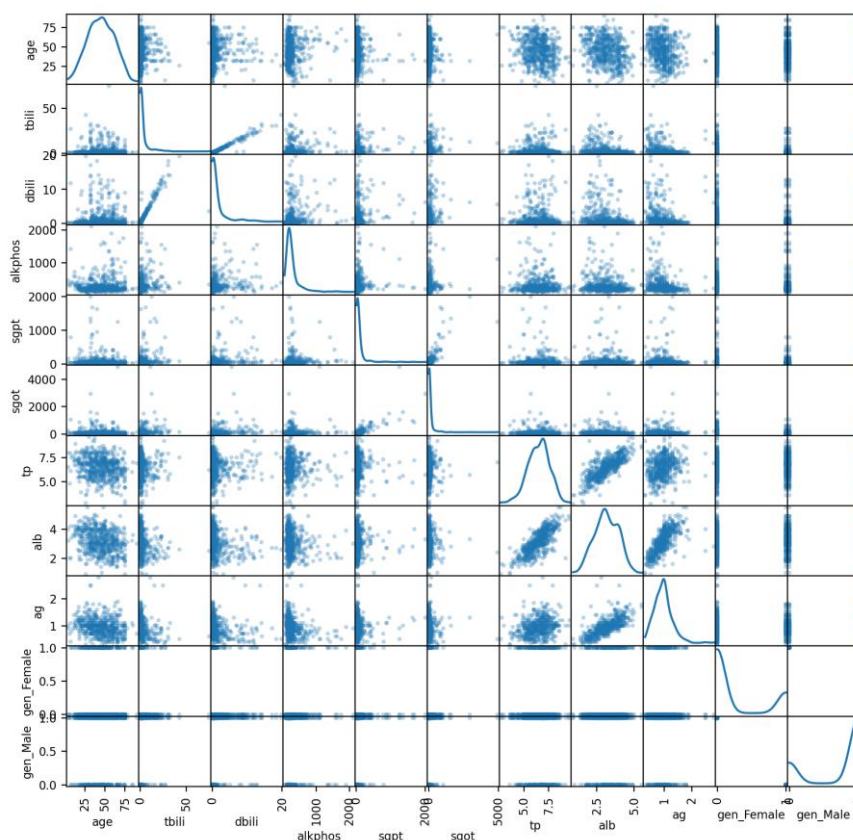


Ilustración 5: Gráfica de la matriz de dispersión del dataset Indian-Liver-Patient.

La Ilustración 4 y la Ilustración 5 han sido obtenidas mediante el siguiente código:

```
from matplotlib import pyplot
from pandas import read_csv
import pandas

filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)

input_data = data[data.columns[:-1]]
input_data = pandas.get_dummies(input_data, prefix=['gen'])

# matriz de correlación
correlations = input_data.corr()

f = pyplot.figure()
cax=pyplot.matshow(correlations, fignum=f.number)
pyplot.xticks(range(correlations.shape[1]), correlations.columns,
rotation=45)
pyplot.yticks(range(correlations.shape[1]), correlations.columns)
f.colorbar(cax, fraction=0.046, pad=0.04)
pyplot.show()

# matriz de dispersión
pandas.plotting.scatter_matrix(input_data, alpha = 0.3, diagonal =
'kde');
```

3. PREPROCESAMIENTO DE DATOS

Una tarea imprescindible a llevar a cabo en todo proyecto de Machine Learning es la preparación de los datos. Se hace necesario puesto que se debe exponer del mejor modo posible la estructura del problema a los algoritmos que se pretenden utilizar. El proceso para preparar los datos en Python se explicará mediante las herramientas que pone a disposición el módulo *scikit-learn*. Concretamente, en esta sección se estudiará cómo:

- | Tratar los datos categóricos.
- | Reescalar los datos.
- | Estandarizar datos.

3.1. Necesidad de preprocesar los datos

El preprocesamiento de los datos es un proceso frecuente y necesario dentro de todo proyecto de Machine Learning. Una dificultad que suele darse es que muchos algoritmos de este tipo realizan suposiciones sobre los datos de entrada, lo que hace que se requieran diferentes transformaciones. Sin embargo, se debe tener en cuenta que, en ocasiones, cuando se siguen todas las reglas y se preparan concienzudamente los datos, los algoritmos ofrecen mejores resultados sin el preprocesamiento.

Por tanto, de modo general, la recomendación sería crear muchas vistas y transformaciones diferentes de los datos, y luego ejecutar el algoritmo con cada una de ellas. De este modo, se podría descubrir qué transformaciones pueden acoplarse mejor a la estructura general del problema.

En el caso específico de las variables categóricas, la mayoría de algoritmos no pueden trabajar con ellas, por lo que es indispensable su tratamiento y transformación.

3.2. Transformaciones de datos

En esta sección se van a estudiar diferentes formas de preprocesamiento de datos para Machine Learning. En cada una se utilizará el conjunto de datos *Indian-Liver-Patient* y todas siguen la misma estructura:

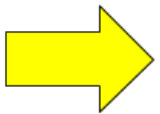
1. Cargar el conjunto de datos.
2. Dividir el conjunto de datos en las variables de entrada y objetivo.
3. Aplicar un preprocesamiento mediante una transformación de las variables de entrada.
4. Mostrar el cambio producido.

Las transformaciones se calculan utilizando los datos de entrenamiento, de tal manera que se genera un modelo a aplicar sobre ellos mismos y se dispone de él para hacer lo mismo sobre cualquier muestra de datos que se pueda tener en el futuro. El proceso llama al método `fit()` del modelo de transformación para preparar los parámetros de la misma utilizando los datos de entrada. Después, se hace uso del método `transform()` sobre los mismos datos para prepararlos para el entrenamiento de los algoritmos. Finalmente, tanto el conjunto de datos de test o validación, como los nuevos datos que pueda verse en el futuro, deben sufrir la misma transformación antes de ser utilizados.

Tratamiento de datos categóricos

Las variables categóricas son datos que sólo toman un número limitado de valores. Sin recodificarlas primero, si se intenta introducir este tipo de variables en la mayoría de los modelos de Machine Learning obtendrá un error en Python. El método más extendido para codificar variables categóricas es el denominado *One-Hot-Encoding*. Este tipo de codificación funciona muy bien a menos que su variable categórica tome un gran número de valores, hecho que provoca que la dimensionalidad de los datos se vea muy afectada. Una codificación *One-Hot-Encoding* crea nuevas columnas (binarias), indicando la presencia de cada valor posible de los datos originales.

Color			
Red			
Red			
Yellow			
Green			
Yellow			



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Ilustración 6: One-Hot-Encoding.

Para implementar esta codificación en Python, Pandas pone a disposición la función `get_dummies()`, que aplica la transformación del DataFrame sobre aquellas variables con *dtype* categórico.

```
from pandas import read_csv
import pandas

filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)
input_data = data[data.columns[:-1]]
print(input_data)
input_data = pandas.get_dummies(input_data, prefix=['g'])
print(input_data)
```

Cambiar la escala de los datos

Cuando los datos se componen de atributos con diferentes escalas, muchos algoritmos pueden verse beneficiados del reescalado de los atributos para que todos pasen a estar dentro del mismo rango. A este proceso se le denomina normalización. A menudo, los atributos se reescalan en el rango entre 0 y 1. Esto resulta de utilidad para los algoritmos de optimización utilizados en el núcleo de los algoritmos Machine Learning, como el descenso de gradiente. También tiene un efecto positivo en los algoritmos que ponderan las entradas, como la regresión y las redes neuronales, y en métodos que utilizan medidas de distancia, como k-NN.

El reescalado de datos se puede implementar mediante la clase *MinMaxScaler* de scikit-learn del siguiente modo:

```
#Reescalado [0,1] datos
from pandas import read_csv
import pandas
from sklearn.preprocessing import MinMaxScaler

filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)

input_data = data[data.columns[:-1]]
input_data = pandas.get_dummies(input_data, prefix=['gen'])
print(input_data.head(5))
X = input_data.values
Y = data['class'].values
minmaxSc = MinMaxScaler(feature_range=(0, 1))
minmaxSc.fit(X)
rescX = minmaxSc.transform(X)
print(rescX[:5,:])
```

Estandarizar los datos

La estandarización es una técnica útil para transformar los atributos con una distribución gaussiana y diferentes medias y desviaciones estándar en una distribución gaussiana estándar con una media de 0 y una desviación estándar de 1. Esta técnica es la más adecuada para los métodos que asumen una distribución gaussiana en las variables de entrada, como la regresión lineal, la regresión logística y el análisis discriminante lineal. Puede estandarizar los datos utilizando scikit-learn con la clase *StandardScaler*.

Ante la presencia de valores atípicos, *StandardScaler* no garantiza escalas de atributos equilibradas debido a la influencia de los valores atípicos al calcular la media y la desviación estándar, estadísticos muy sensibles a los outliers. Esto lleva a la concentración del rango de los valores de los atributos. Utilizando *RobustScaler* de scikit-learn, se pueden eliminar los valores atípicos y luego utilizar *StandardScaler* o *MinMaxScaler* para el preprocesamiento del conjunto de datos.

El *RobustScaler* escalará los atributos utilizando estadísticos que son robustos a los valores atípicos. Concretamente, este método resta a cada valor la mediana de la variable y escala los datos en el rango intercuartílico.

```
from pandas import read_csv
import pandas
from sklearn.preprocessing import StandardScaler, RobustScaler

filename = 'Indian-Liver-Patient.csv'

col_names = ['age', 'gen', 'tbili', 'dbili', 'alkphos', 'sgpt',
             'sgot', 'tp', 'alb', 'ag', 'class']
data = read_csv(filename, names=col_names)

input_data = data[data.columns[:-1]]
input_data = pandas.get_dummies(input_data, prefix=['gen'])
print(input_data.head(5))
X = input_data
Y = data['class']

stdScaler = StandardScaler().fit(X)
rescX = stdScaler.transform(X)
print(rescX[:5,:])

robScaler = RobustScaler().fit(X)
rescX = robScaler.transform(X)
print(rescX[:5,:])
```

4. PUNTOS CLAVE

En esta lección hemos aprendido:

- | Sumergirse en los datos a través de Python para obtener el máximo rendimiento de ellos.
- | Implementar las técnicas de estadística descriptiva para resumir los datos numérica y gráficamente.
- | Utilizar las herramientas de Pandas para analizar las relaciones presentes en los datos, numérica y gráficamente.
- | Conocer los principios y saber aplicar las técnicas de preprocesamiento de datos con Pandas y scikit-learn.

